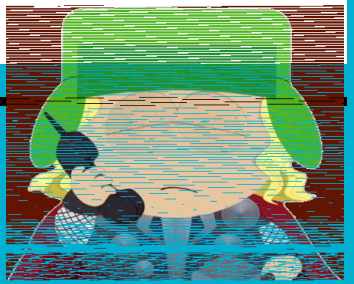
UA05b. Json

Ejercicios

Ejercicio 1:
Creación simple



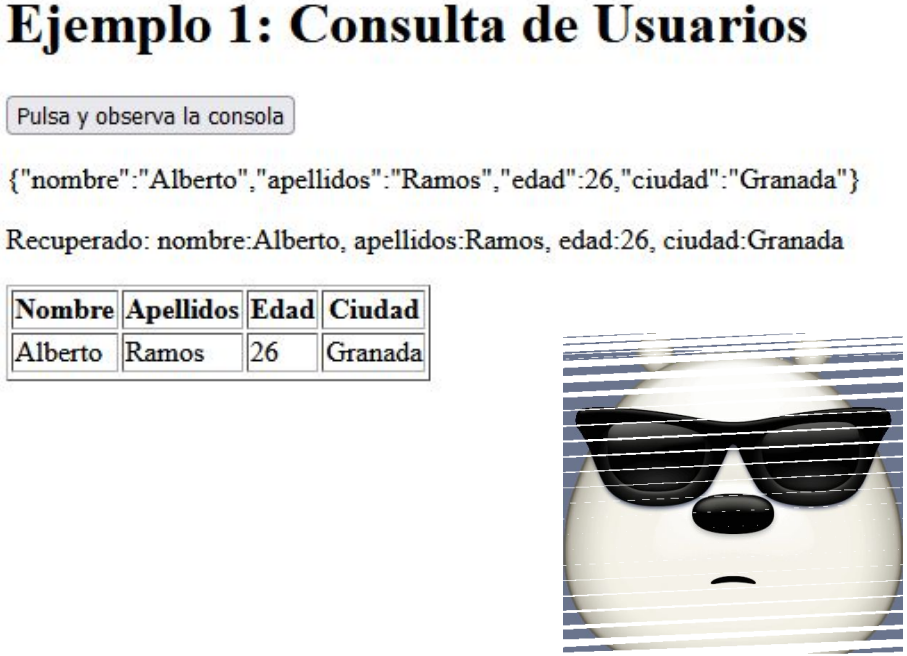
Ejercicio 1: Creación simple

Crea un archivo **HTML con Javascript** que convierta un objeto Javascript en JSON y viceversa (serialización-deserializacion). Deberás crear una función para cada acción.

Muestra ambos resultados por consola usando **console.log** o mediante elementos HTML como **div**, **p** o **span**.

Recoge datos personales: **nombre**, **apellidos**, **edad y ciudad.**

Guarda el ejercicio como ej1.html

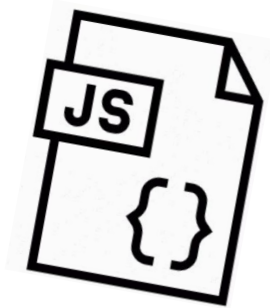


Ejercicio 1: Ejemplo

```
let objJson = {nombre: "Alberto",apellidos:"García",edad: 26,ciudad: "Granada"};

json_serializado = serializar(objJson);

console.log(json_serializado);

obj_recuperado = deserializar(json_serializado);

console.log(obj_recuperado);

function serializar(obj){

      return JSON.stringify(obj)

}

function deserializar(objJson){

      return JSON.parse(objJson)

}
```

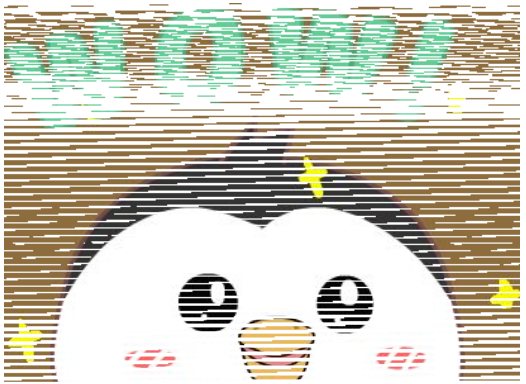
Ejercicio 1: Solución JavaScript

```
let texto = "Nombre:" + objeto_recuperado.nombre;

texto += ", Apellidos:" + objeto_recuperado.apellidos;

texto += ", Edad:" + objeto_recuperado.edad;

texto += ", Ciudad:" + objeto_recuperado.ciudad;

console.log("Recuperado: " + texto);

document.getElementById("deserializado").innerHTML = "Recuperado: " + texto;
```



Ejercicio 1: Solución Jquery

```
let texto = "Nombre:" + objeto_recuperado.nombre;

texto += ", Apellidos:" + objeto_recuperado.apellidos;

texto += ", Edad:" + objeto_recuperado.edad;

texto += ", Ciudad:" + objeto_recuperado.ciudad;

console.log("Recuperado: " + texto);

$("#deserializado").text("Recuperado: " + texto);
```



Ejercicio 1: RETO

Crea una página HTML con una Tabla que contenga una zona **thead** con los nombres de los datos que se mostrarán, y una zona **tbody** con los datos recuperados tras la serialización-deserialización.

Usa identificadores para insertar los datos en las celdas de la tabla desde Javascript.

Prueba con **Jquery**.

Nombre	Apellidos	Edad	Ciudad
Alberto	Ramos	26	Granada

Guarda el ejercicio como ej1reto.html

Ejercicio 2:
Insertar datos dinámicos

Ejercicio 2: Insertar datos dinámicos

Basándonos en el ejercicio anterior, crea un pequeño **formulario** que recoja la información anterior para luego adjuntarla (**append**) a la tabla en una nueva fila.

La página no debe recargarse en cada inserción.

Para realizar esto, una **nueva función** devolverá una cadena con una fila TR y las celdas TD necesarias con los datos insertados provenientes del **Json deserializado** (argumento de entrada).

El retorno de esta función se usará para adjuntarlo al **tbody** de la tabla.

Guarda el ejercicio como ej2.html

Ejercicio 2: Ejemplo

```
function generaTr(objeto_recuperado) {
     return "<tr>”+
     “<td>" + objeto_recuperado.nombre + "</td>”+
     “<td>" + objeto_recuperado.apellidos + "</td>”+
     “<td>" + objeto_recuperado.edad + "</td>”+
     “<td>" + objeto_recuperado.ciudad + "</td>”+
     “</tr>";
}
```

Ejercicio 2: Solución

```
let objJson = {
     id: $(“#id”), nombre: $("#nombre").val(), apellidos: $("#apellidos").val(),
     edad: $("#edad").val(), ciudad: $("#ciudad").val(),
};
// Objeto JavaScript a JSON
const json_serializado = serializar(objJson);
// JSON a objeto JavaScript
const objeto_recuperado = deserializar(json_serializado);
$("#tbody").append(generaTr(objeto_recuperado));
```

Ejercicio 2: RETO

Comprueba que todos los campos estén rellenos. En caso contrario, deberá mostrarse una alerta (**alert**) indicando que **el campo xxx es obligatorio**.

Añade la acción **onclick** al TR para que al pulsar se elimine la fila de la lista. Elimina el TR con **remove**. Usa el id.

Muestra en un elemento adicional (**p**, **div**, **span**, …) o en consola (**console.log**) la suma de edades, media, máximo y mínimo.

```
Suma = 120          Máximo = 35

Media = 30          Mínimo = 15
```

Guarda el ejercicio como ej2reto.html

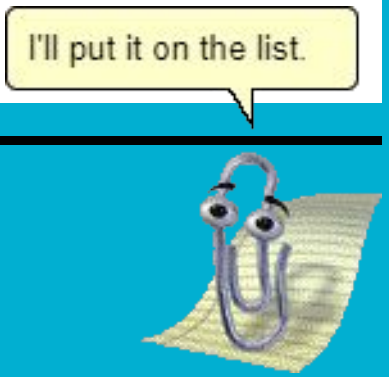
Ejercicio 2: MEGARETO

Modifica el apartado anterior para realizar lo siguiente:

1. Crea una array en JavaScript que guarde las filas que se van insertando.
2. Inicialmente (al cargar la página), esta variable estará vacía.
3. Usa una función para añadir (**push**) los datos del formulario al objeto en cada inserción.
4. Serializa la lista a Json.
5. Pasa esta variable a la función **generaTabla** donde, tras deserializarla, se eliminará y creará la tabla. Por cada elemento del objeto recuperado (**forEach**), se generará un TR con la función creada en diapositivas anteriores (**generaTR**).
6. Añade el reto anterior pero, **al eliminar**, después (o antes) del TR deberá desaparecer la fila del **OBJETO**. Usa **splice** para ello.
7. Para el borrado del objeto, puede ser útil añadir un **campo id**.

Guarda el ejercicio como ej2mega.html

Ejercicio 3: Listas



Ejercicio 3: Listas

Modifica el formulario anterior e inserta un grupo de checkboxes para indicar alguno de los siguientes **hobbies** para cada usuario insertado:

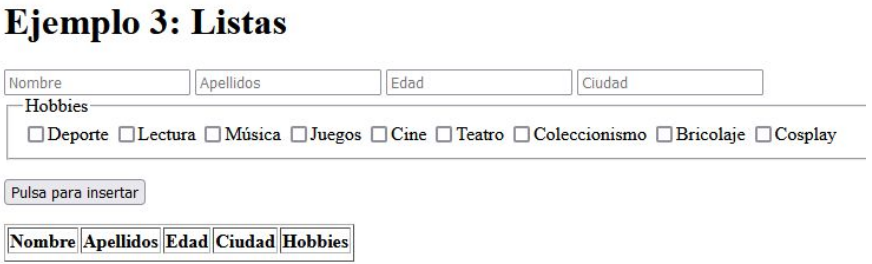
- deporte,
- lectura,
- música,
- juegos,
- cine,
- teatro,
- coleccionismo,
- bricolaje
- cosplay.

Agrega una nueva columna a la tabla para mostrar los hobbies seleccionados.

Guarda el ejercicio como ej3.html

Para la lectura de las casillas marcadas, puedes hacer uso de algo como esto:

```
hobbies: $('input[name="hobbies"]')
     .filter(':checked')
     .map(function () {
          return $(this).val();
     }
).get()

// get() convierte el objeto en array
```



Ejercicio 3: RETO

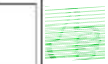
Lola | Mento | 24 | Murcia

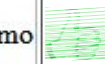
Hobbies

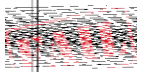
☑ Deporte ☑ Lectura ☑ Música ☐ Juegos ☑ Cine ☑ Teatro ☑ Coleccionismo ☑ Bricolaje ☑ Cosplay

Pulsa para insertar

Nombre	Apellidos	Edad	Ciudad	Hobbies
John	Rambo	44	Lepe	Música Cine
Mary	Austin	38	Pilas	Teatro Coleccionismo Cosplay
Lola	Mento	24	Murcia	Deporte Lectura Música Cine Teatro Coleccionismo Bricolaje Cosplay







Vamos a limitar la opción de elegir hobbies a un **máximo de tres de ellos**.

No se desea comprobar cuántos se han elegido al pulsar el botón, sino que, mientras se selecciona, evitar que se marquen más de tres.

Para ello, puedes usar Jquery indicando el elemento **input:checkbox**.

Guarda el ejercicio como ej3reto.html

Ejercicio 3: Pistas

La siguiente acción comprueba si se ha pulsado un checkbox:

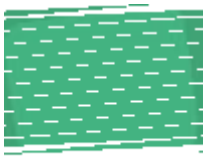
```
$('input:checkbox').on('change', function () { /* TO-DO */ }
```

La siguiente acción devuelve la cantidad de checkboxes marcados:

```
$('input:checkbox').filter(':checked').length
```

La siguiente acción cambia un atributo de un elemento mediante **prop**:

```
$(this).prop('checked', true);

$(this).prop('disabled', false);
```



Ejercicio 3: Solución

```
<script>
    // Variables globales para los cálculos
    // Variable global para el array de objetos
    // Acción con Jquery que chequea las casillas pulsadas para que sean < 3
    // Función Principal
        // Compruebo campos vacíos
        // Inserto valores leídos de los campos y casillas
        // Serializo
        // Genero Tabla
    // Función Insertar(objeto)
        // Realizo cálculos
        // Inserto objeto en array
    // Función generaTabla(objeto)
        // Por cada fila, genero un TR
    // Función generaTR(objeto)
    // Función eliminarTR(TR)
</script>
```

Ejercicio 4:
Envío y recepción

Envío y recepción

Vamos a realizar dos acciones:

1. Recibir datos de un servidor
2. Enviar datos a un servidor

Hasta ahora, conocemos envíos GET y POST mediante formulario HTML. Estos métodos no son dinámicos ya que se cambia de página cuando se realiza el envío.

Vamos a aprender a enviar y recibir datos usando técnicas dinámicas como:

- XMLHttpRequest
- AJAX
- Jquery

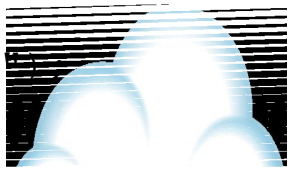
XMLHttpRequest

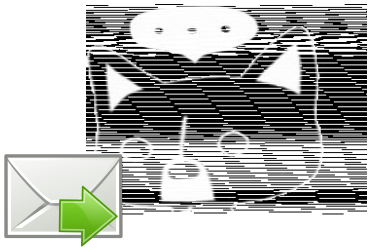
Para completar una petición AJAX, se realizan las siguientes acciones:

1. Crear un objeto XMLHttpRequest
2. Abrir la conexión al URL en modo GET o POST (aunque hay otras)
   a. GET para lecturas y recepción de datos
   b. POST para escrituras y envío de datos
3. Se indica el tipo de respuesta que se espera
4. Se prepara la comprobación de la respuesta:
   a. si el estado es 200, se captura la respuesta (**response**)
   b. si no, podemos gestionar el error
5. Se realiza el envío

Ejemplo de recepción con XMLHttpRequest

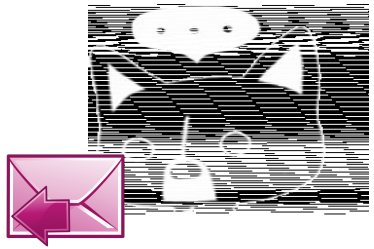
```
function solicitud(){
    const xhr = new XMLHttpRequest();
    xhr.open("GET", "https://jsonplaceholder.typicode.com/users");
    xhr.responseType = "json"; // Si no se indica, necesitará parseo

// Preparamos a continuación la recepción
    xhr.onload = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            const data = xhr.response;
            console.log(data);
        } else {
            console.log("Error: ${xhr.status}");
        }
    };
    xhr.send();
}
```





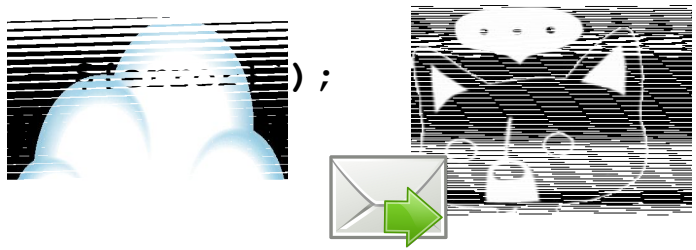
Ejemplo de envío con XMLHttpRequest

```
function envio(objeto_js){
    const xhr = new XMLHttpRequest();
    xhr.open("POST", "https://jsonplaceholder.typicode.com/posts");
    xhr.responseType = "json"; // Si no se indica, necesitará parseo

// Preparamos a continuación la respuesta
    xhr.onload = function() {
        if (xhr.readyState == 4 && xhr.status == 201) { // 200 || 201
            console.log(xhr.response);
        } else {
            console.log("Error: ${xhr.status}");
        }
    };
    xhr.send(JSON.stringify(objeto_js));
}
```



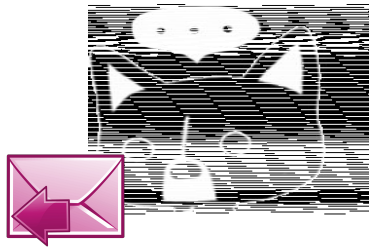
Ejemplo de recepción con AJAX

```
function solicitud() {
    $.ajax({
        url: "https://jsonplaceholder.typicode.com/users",
        method: "GET",
        dataType: "json", // Convierte la respuesta a objeto JSON
        success: function(data) { //200 o 201
            console.log(data);
        },
        error: function(xhr, status, error) {
            console.log(`Error: ${xhr.status}                    );
        }
    });
}
```



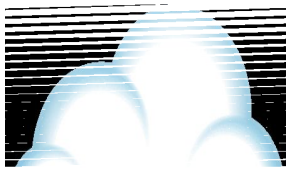
Ejemplo de envío con AJAX

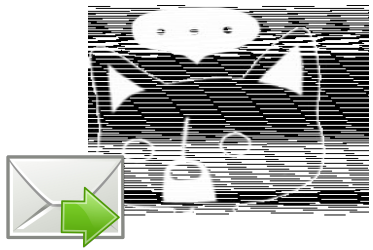
```
function envio(objeto_js) {
    $.ajax({
        url: "https://jsonplaceholder.typicode.com/posts",
        method: "POST",
        data: JSON.stringify(objeto_js),
        contentType: "application/json", // Especifica el tipo de contenido
        dataType: "json", // La respuesta será interpretada como JSON
        success: function(response) {
            console.log(response);
        },
        error: function(xhr, status, error) {
            console.log(`Error: ${xhr.status} - ${error}`);
        }
    });
}
```



Ejemplo de recepción con JQuery

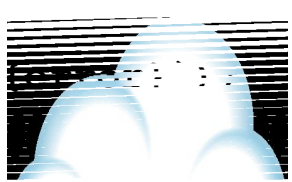
```
function solicitud() {
    $.get("https://jsonplaceholder.typicode.com/users")
        .done(function(data) {
            console.log(data);
        })
        .fail(function(xhr, status, error) {
            console.log(`Error: ${xhr.status} - ${error}`);
        });
}
```

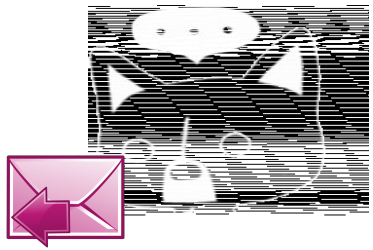




Ejemplo de envío con JQuery

```
function envio(objeto_js) {
    $.post("https://jsonplaceholder.typicode.com/posts",
        JSON.stringify(objeto_js),
        function(response) {
            console.log(response);
        },
        "json"
    )
    .fail(function(xhr, status, error) {
        console.log(`Error: ${xhr.status} - $
    });
}
```





Ejercicio 4: Envío y recepción

Modifica el ejercicio 3 para realizar una prueba de envío y recepción a la página **https://lm.iesnervion.es/eco.php**. Utiliza las funciones anteriores.

Esta dirección URL debe recibir un array JSON con la lista actual más los datos del formulario para procesarlos. Posteriormente, serán devueltos junto con los cálculos realizados: suma, media, máx y min.

Observa la respuesta para saber qué hacer.

Elige el método que quieras para realizarlo.

Guarda el ejercicio como ej4.html

Ejercicio 4: RETO

La dirección web **https://lm.iesnervion.es/reto4.php** espera recibir datos para insertarlos en una tabla de Base de Datos MariaDB. La sentencia para la construcción de dicha tabla fue:

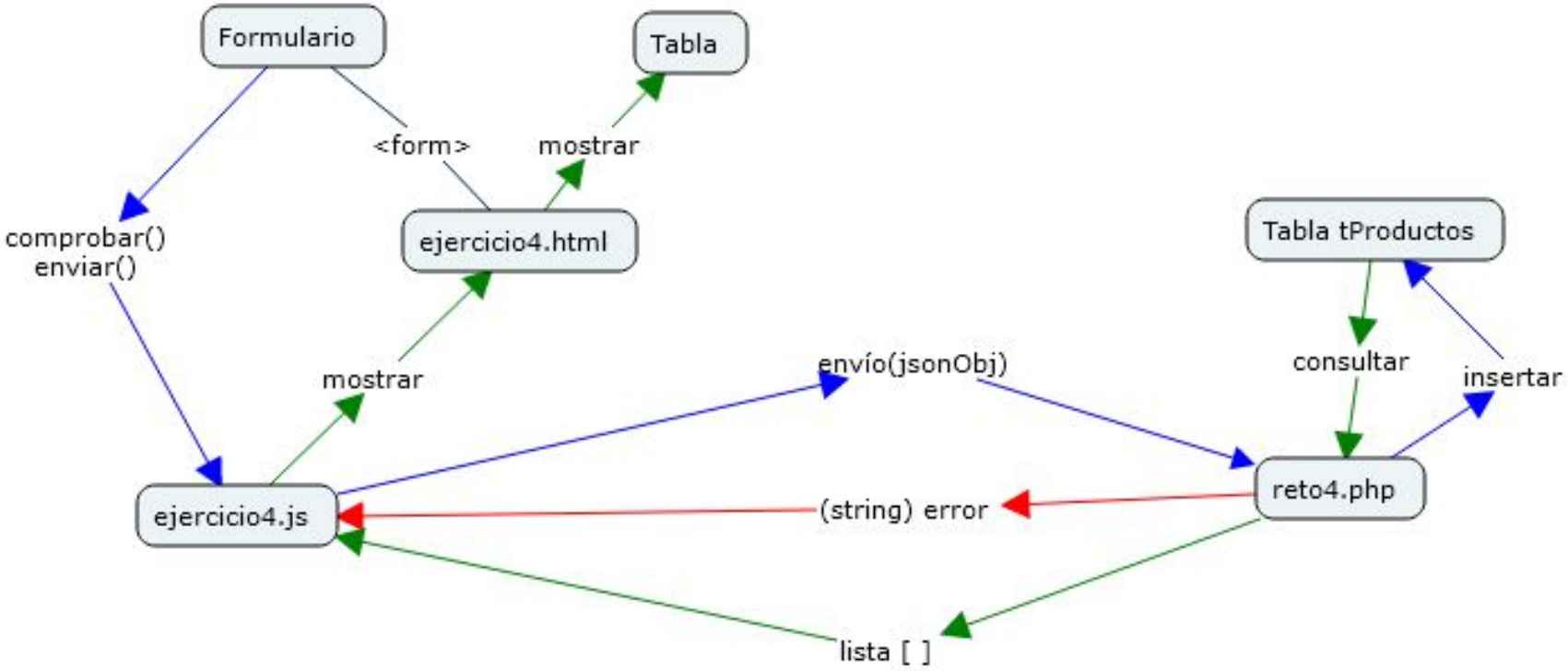
```
CREATE TABLE tProductos (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    descripcion VARCHAR (255),
    idProveedor BIGINT,
    precio DECIMAL (5,2),
);
```

Crea un archivo HTML con JS que recoja la información que necesita la base de datos y la envíe al servidor. Éste último devolverá una **lista** y un **error** (usa la consola).

La lista obtenida **deberá mostrarse en una tabla** de productos en la propia página y el **error** en un **h2** en **negrita y rojo**.

Si el error es NULL indicará que todo fue OK. El siguiente esquema te será útil.

Esquema Reto 4



Ejercicio 4: MEGARETO

Crea una copia del ejercicio anterior y modifica lo necesario para realizar lo siguiente:

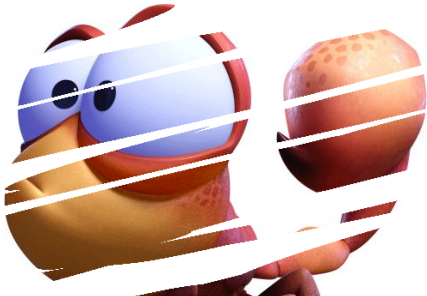
Con la respuesta obtenida al insertar, usa el campo **id** como identificador de la fila TR.

De esta forma, al pulsar sobre el TR, enviaremos este identificador al servidor con el método DELETE para indicar que se desea eliminar esa línea de la base de datos.

Usa este método en lugar de los anteriores:

```
xhr.open("DELETE", url+"?"+nombreVariable+"=" + valorVariable, true);
```

Si no devuelve errores, muestra el mensaje "**Se ha eliminado correctamente**" en verde, dentro del elemento h2 del RETO anterior.



Ejercicio 5:
Conexión a Base de Datos

Las Bases de Datos

Las aplicaciones y las webs que diseñamos usan información almacenada en bases de datos, ya sean locales o remotas.

Durante el curso se ha trabajado con Bases de Datos en modo local (*localhost*), pero sólo tiene interés si se está montando un servidor para conexión externa o se usa para pruebas en modo local.

Ya conocemos la manera de instalar y gestionar una base de datos.

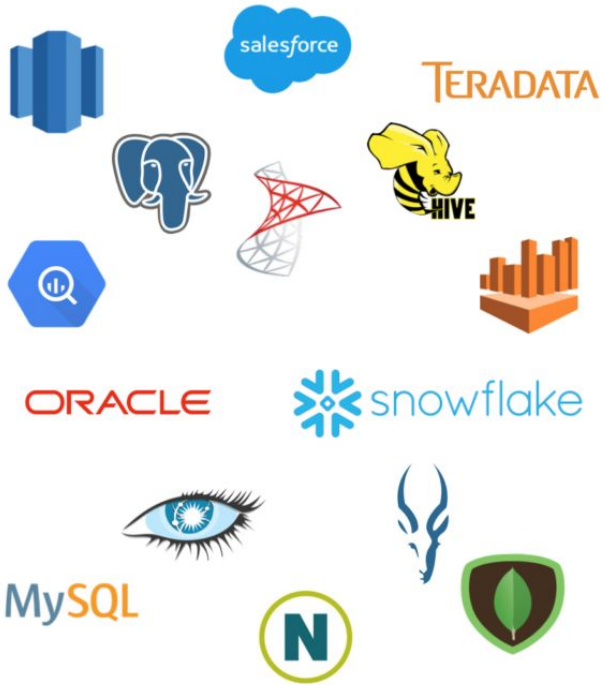
Vamos ahora a acceder a ellas.

La conexión a la Base de Datos

Desde Javascript NO es posible conectarse directamente a una base de datos.

La conexión a la base de datos se realiza de forma remota y necesitaremos un controlador (*driver*) para manejarla.

Los diferentes servidores de Bases de Datos (MySQL, PostgreSql, Oracle, SQL Server, …) disponen de controladores para todos o la gran mayoría de lenguajes de programación.



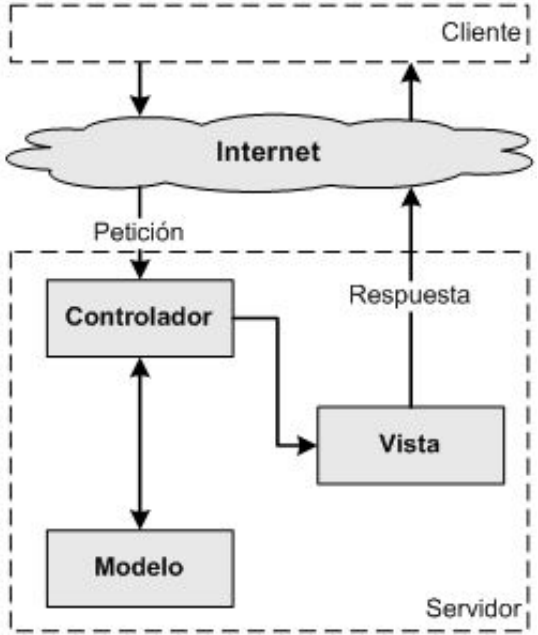
La conexión a la Base de Datos

Para ello, se utilizan como intermediarios los lenguajes del lado del servidor (por ejemplo, PHP o NodeJS) realizando peticiones (**request**) a las URLs utilizadas como controladores (denominados **endpoints**).

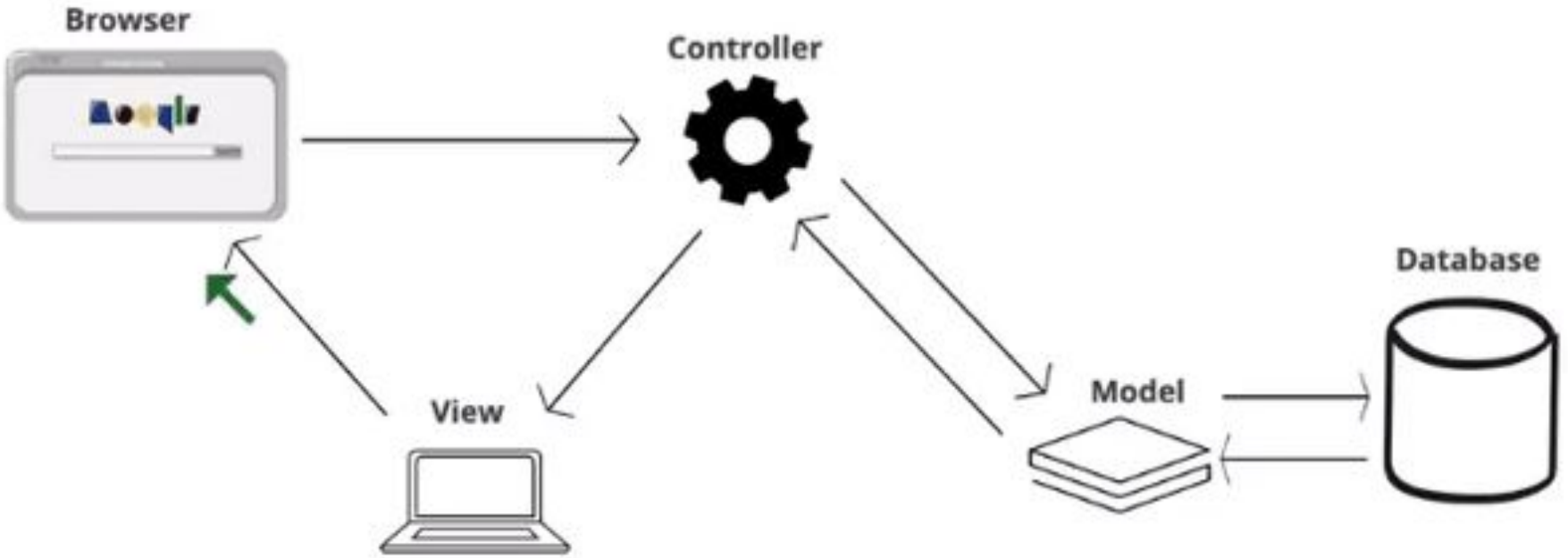
Un endpoint utilizará los modelos como intermediarios para acceder a la base de datos. Concretamente a las tablas y el contenido de éstas.

Finalmente, devolverá una respuesta (**response**) con los datos que se mostrarán en algún elemento de la página u otra página a través de las vistas.

Este patrón de diseño es el conocido como Modelo-Vista-Controlador aunque existen otros **patrones MVC**, aunque existen más modelos.



Esquema funcional



Ejemplo: La clase Modelo (lado Servidor)

```php
<?php
class Modelo {
   private $conexion;
   public function   construct(){
       $this->conexion = new mysqli(
          "localhost",
          "usuario",
          "contraseña",
          "nombre base de datos");
       if (!$this->conexion) {
           die("Error de conexión");
       }
   } // Fin constructor
```

```php
public function obtenerDatos(){
   $sql = "SELECT * FROM nombre_tabla";
   $resultado = $this->conexion->query($sql);
   if ($resultado->num_rows > 0) {
     return $resultado->fetch_all(MYSQLI_ASSOC);
   } else {
// Devuelve un array vacío si no hay resultados
      return array();
   }
} // Fin obtenerDatos

} // Fin clase
```

Ejemplo: el controlador (lado Servidor)

```php
<?php
// controlador.php
include_once 'modelo.php';  // La clase Modelo

$modelo = new Modelo();     // Se crea el objeto
$datos = $modelo->obtenerDatos();

include 'vista.php';        // Vista HTML para mostrar
```

Ejemplo: la vista (lado Servidor)

```
<!-- vista.php -->
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mostrar Datos</title>
</head>
<body>
<h1>Datos obtenidos desde la base de datos:</h1>
<ul>
    <?php foreach ($datos as $dato){    // Bucle para recorrer la lista ?>
        <li><?php echo $dato['campo']; // Print del item de lista ?></li>
    <?php }                             // Fin del bucle ?>
</ul>
</body>
</html>
```

Ejemplo usando HTML, CSS y JS

Ejemplo MVC usando HTML, CSS y Javascript

Modelo:

```
// model.js
const Model = {
  data: {
    items: [],
  },
  addItem(item) {
    this.data.items.push(item);
  },
  getItems() {
    return this.data.items;
  },
};
```

Ejemplo MVC usando HTML, CSS y Javascript

Vista:

```
// view.js
const View = {
  render(items) {
    const $list = $('#item-list');
    $list.empty(); // Limpia la lista
    items.forEach((item) => {
      $list.append(`<li>${item}</li>`);
    });
  },
};
```

En el HTML debe existir algo como esto: <ul id="item-list">

Ejemplo MVC usando HTML, CSS y Javascript

Controlador:

```
// controller.js
const Controller = {
 init() {
   this.bindEvents();
   this.updateView();
 },
 bindEvents() {
   $('#add-item-button').on('click',
function(){
   const newItem=$('#item-input').val();
      if (newItem) {
        Model.addItem(newItem);
        this.updateView();
      }
    });
  },
  updateView() {
    const items = Model.getItems();
    View.render(items);
  },
};
```

En el HTML puede haber algo como esto:

```
<input type="text" id="item-input" placeholder="Insertar un item">
<button id="add-item-button">Insertar</button>
```

Organización e index.html

```
/project
  index.html
  /css
    styles.css
  /js
    model.js
    view.js
    controller.js
```

```
  <link rel="stylesheet" href="css/styles.css">

  <script
src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script src="js/model.js"></script>
  <script src="js/view.js"></script>
  <script src="js/controller.js"></script>

<body>
  <input type="text" id="item-input" placeholder="Add an item">
  <button id="add-item-button">Add</button>
  <ul id="item-list"></ul>

  <script>
    $(document).ready(() => {
      Controller.init();
    });
  </script>
</body>
```